Just-in-time execution is not always optimal!

Twan Basten[1,2], Marc Geilen[1], Sander Stuijk[1]

[1]Eindhoven University of Technology
[2]TNO-ESI, Embedded Systems Innovation by TNO

***Abstract*—Synchronous Data Flow (SDF) is a popular model for analyzing workflows. It is well-known that self-timed scheduling, starting a task when all its prerequisites are satisfied, is throughput-optimal for SDF graphs. Just-in-time scheduling in the presence of deadlines has so far received little attention in the literature. We show that a feasible just-in-time schedule exists if and only if a feasible self-timed schedule exists. Consequently, just-in-time execution is also throughput-optimal. It is, however, not necessarily optimal when tasks may occasionally take longer than anticipated. We experimentally conclude that the likelihood of missing deadlines is significantly higher, 16%, than it is for self-timed scheduling. This shows that Ralph Otten's way of working, just-in-time scheduling without exception, is not optimal.**

I. Workflows

In this paper, we consider the scheduling of workflows consisting of repeating tasks with dependencies and deadlines. Two special cases of scheduling are considered. On the one hand, *self-timed* or *data-driven* scheduling in which tasks are executed as soon as enabling conditions are satisfied, and on the other hand *just-in-time* scheduling where the tasks are started soon enough to meet the deadlines, but not sooner.

As the basis for our investigation we use the Synchronous Data Flow (SDF) [5] model of computation to represent such workflows. Fig. 1(a) shows an example of a Synchronous Data Flow Graph (SDFG) that may abstractly model a certain periodically occurring workflow that involves multiple tasks with dependencies. The circles in the graph are the *actors* that represent the work that is to be done for individual tasks. Such work can start only when all dependencies of the task have been met. The execution of an actor is called a *firing*. The firing condition is modelled by all incoming edges (arrows) of the actor having sufficient *tokens* indicated by dots along the edges. In this example, a single token on each edge is required, but other rates may be specified. The actors are further annotated with an *estimated execution time*. For instance, the Src actor takes 6 time units to complete. Actors may repeatedly fire, even multiple times concurrently. Immediately after an actor finishes, it *produces* tokens on the outgoing edges. In the example, all actors produce one token on each edge, but in general other rates are possible. The periodic occurrence of work to perform is modeled by the actor named Src. The only incoming dependency comes from itself with one token on the dependency. This implies that one firing of Src needs to complete before the next one can start. Execution of the workflow proceeds with the firings of actors A, B and C which have mutual dependencies. The execution of a single instance of the workflow ends in the firing of the Snk actor. The completion time of the workflow instance is modeled by the moment that the corresponding token is produced on the edge from the Snk actor to itself. Using classical analysis techniques for SDF [5], [3] we can demonstrate that the workflow tasks between Src and Snk can be repetitively executed at a period as small as 5 time units. In this case, the source releases instances of the workflow at a lower rate with at least 6 time units between instances, modelled with the execution time of 6 for Src and Snk.

With the constraints from dependencies and execution times of the SDFG, there is still freedom to consider alternative schedules. Fig. 1(c) shows an example of a schedule for the graph in the form of a Gantt chart. In this particular schedule, all actors start their firings as soon as possible, which gives the self-timed schedule. The circled numbers k indicate the production times of the k-th token at the different initial token locations in the graph. They represent both the enabling times for the actor firings that consume these tokens and the completion times of the actors that produce them. We further constrain the set of admissible schedules by assuming that deadlines are set to the firings of the Snk actor. In Fig. 1(c), the deadlines are indicated by the $\times$ symbols; deadlines are set to $11 + n \cdot 6$ for the n-th execution of Snk, starting from $n = 1$. The self-timed schedule meets its deadlines.

An alternative schedule, also meeting its deadlines is depicted in Fig. 1(d). In this particular schedule, all actor firings are scheduled just in time. Any increase in start time inevitably leads to a violation of a deadline. The big question addressed in this work is which kind of schedule is to be preferred? We provide the answer. Even though both self-timed and just-in-time schedules deliver the same throughput and meet their deadlines when all tasks proceed according to plan, a self-timed schedule is more robust to incidental excesses of the estimated execution times of tasks. It is less likely to miss deadlines. Hence, self-timed scheduling is preferred; just-in-time execution is not optimal.

We proceed as follows. We define what a just-in-time schedule is and show that the just-in-time schedule of an SDFG is unique. We also show how it can be effectively computed. To establish this computation, we prove the correspondence between the just-in-time execution of an SDFG and the self-timed execution of the related SDFG obtained from the original one by reversing all the edges in the graph. We prove that feasibility of both schedules coincides, that they have the same throughput and that self-timed execution is strictly better in terms of deadline misses under occasional violations of the execution time estimations. We experimentally quantify to what extent just-in-time execution experiences more deadline misses than self-timed execution.

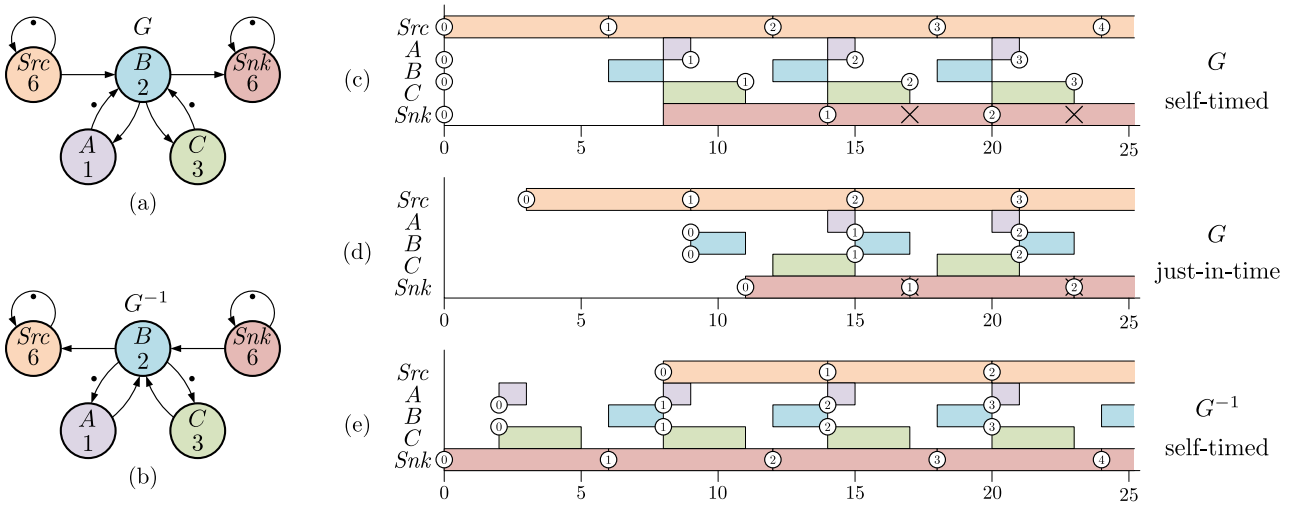

Fig. 1. An example SDF model of a workflow graph, its reversed graph and different schedules

II. Analysis

SDF is amenable to powerful analysis, synthesis and scheduling techniques [6], [5], [3]. We can efficiently analyse throughput and latency constraints of an SDFG and determine its resource requirements. Moreover, we can effectively generate schedules according to different scheduling strategies. The self-timed schedule in Fig. 1(c) for the example SDFG is well known [6], [3] to lead to maximal throughput.

Due to the constant rates of the dependencies, SDFGs execute in repetitive groups of actor firings called *iterations*. In the simple example of Fig. 1, an iteration consists of a single firing of each of the actors. The special property of an iteration is that after all firings in an iteration, the distribution of tokens on edges returns to its original configuration.

Analysis of timed dataflow models can be conveniently done in the linear system theory of $(\max, +)$-algebra [1], [4]. In this context, it suffices to say that $(\max, +)$-algebra defines a linear algebra of matrices and vectors analogous to traditional linear algebra, but with the usual addition operation replaced by the $\max$ operation and the usual multiplication replaced by addition. These operations are used on the real numbers $\mathbf{R}$ extended with $-\infty$ and using the convention that for all $x \in \mathbf{R} \cup \{-\infty\}$, $x \max -\infty = -\infty \max x = x$ and $x + -\infty = -\infty + x = -\infty$. In particular, with this algebra, it is possible to compute for a given SDFG G, a $(\max, +)$-matrix $\mathbf{G}$, called the *characteristic matrix* of G, which exactly characterizes the timing constraints expressed by the model G in terms of the tokens in the graph across a single iteration of G as follows.

$$\boldsymbol{\gamma}' = \mathbf{G}\boldsymbol{\gamma}$$

Vector $\boldsymbol{\gamma}$ has an entry for each of the tokens in the SDFG. A value in the vector represents the enabling time of the dependency captured by the token at the *start of the iteration*. $\boldsymbol{\gamma}'$ then similarly captures the enabling times of token dependencies, but now *at the end of the iteration*. It is well-known [2] how to compute this matrix for a given SDFG and there are tools such as SDF^3 [8] that can compute it. For the example graphs in Fig. 1(a) and (b), the matrices are given below.

$$\mathbf{G} = \begin{pmatrix} 6 & -\infty & -\infty & -\infty \\ 9 & 3 & 3 & -\infty \\ 11 & 5 & 5 & -\infty \\ 14 & 8 & 8 & 6 \end{pmatrix}$$

$$\mathbf{G}^T = \begin{pmatrix} 6 & 9 & 11 & 14 \\ -\infty & 3 & 5 & 8 \\ -\infty & 3 & 5 & 8 \\ -\infty & -\infty & -\infty & 6 \end{pmatrix}$$

Suppose that initially all token dependencies are enabled at time 0. This can be expressed by the vector $\boldsymbol{\gamma}_0 = [0\ 0\ 0\ 0]^T$. We can compute that after a complete iteration the new tokens will be as follows.

$$\boldsymbol{\gamma}_1 = \mathbf{G}\boldsymbol{\gamma}_0 = [6\ 9\ 11\ 14]^T$$

We can verify this in the self-timed schedule in Fig. 1(c), where $\boldsymbol{\gamma}_0$ is represented by the encircled 0s and $\boldsymbol{\gamma}_1$ by the encircled 1s. (The values in the vector correspond to the position of the tokens on the horizontal time axis.)

If we continue, we find (in agreement with Fig. 1(c)) that

$$\boldsymbol{\gamma}_2 = \mathbf{G}\boldsymbol{\gamma}_1 = [12\ 15\ 17\ 20]^T, \quad \boldsymbol{\gamma}_3 = \mathbf{G}\boldsymbol{\gamma}_2 = [18\ 21\ 23\ 26]^T$$

and so on. The matrix $\mathbf{G}$ captures an entire iteration of SDFG G in one matrix multiplication.

Matrix $\mathbf{G}$ also reveals the maximal throughput or minimal period of the SDFG. Its eigenvalue equals the minimal period per iteration. For the example, the eigenvalue of $\mathbf{G}$ is 6, representing a throughput of $1/6$ iterations/time unit.

III. Schedules

We have seen that $(\max, +)$-multiplication can be used to compute the self-timed schedule of an SDFG. But what about a just-in-time schedule, as the one in Fig. 1(d)? First, we need a notion of deadlines for workflows. For our example, we require that actor *Snk* fires every 6 time units starting from time 17. The $\times$ symbols in Fig. 1(c) and (d) represent these deadlines. Note that the completion times of the firings of *Snk*

are recorded in the token on its self edge. We can therefore express the deadlines as upper bounds on the vectors $\boldsymbol{\gamma}_k$. We define a deadline vector $\boldsymbol{d}(n) = [\infty\ \infty\ \infty\ 11 + n \cdot 6]^T$ and require that the following holds for all n.

$$\boldsymbol{\gamma}_n \preceq \boldsymbol{d}(n)$$

(For $(\max, +)$-vectors we say that $\boldsymbol{\alpha} \preceq \boldsymbol{\beta}$ if and only if for all i, $\boldsymbol{\alpha}(i) \leq \boldsymbol{\beta}(i)$; a deadline of ∞ is used to indicate that there is no deadline specified on such token.)

We show that we can effectively compute the deadlines for producing the tokens for the start of an iteration (and hence for the end of the previous iteration) using the matrix $\mathbf{G}^T$ and the following equation ($(-\boldsymbol{\alpha})(i) = -(\boldsymbol{\alpha}(i))$).

$$\boldsymbol{\gamma}_{k-1} = -\mathbf{G}^T(-\boldsymbol{\gamma}_k) \tag{1}$$

With $\boldsymbol{\gamma}_k = [\infty\ \infty\ \infty\ 11 + k \cdot 6]$, we get from Eq. 1:

$$\boldsymbol{\gamma}_{k-1} = [-3 + k \cdot 6;\ 3 + k \cdot 6;\ 3 + k \cdot 6;\ 5 + k \cdot 6]^T = [3\ 9\ 9\ 11]^T + (k-1) \cdot 6$$

Note that now all tokens have deadlines to allow the sink actor to make its deadline. Continuing to compute with these earlier deadlines, we get:

$$\boldsymbol{\gamma}_{k-2} = [-3\ 3\ 3\ 5]^T + (k-1) \cdot 6 = [3\ 9\ 9\ 11]^T + (k-2) \cdot 6$$

Vector $\boldsymbol{\gamma}_{k-2}$ is equal to $\boldsymbol{\gamma}_{k-1}$ except for a shift of all deadlines by the period of 6 and we know that the same will happen for all deadlines on earlier iterations. So, if the deadline needs to be met for all n, the deadlines for all tokens are as follows:

$$\boldsymbol{\gamma}_n = [3\ 9\ 9\ 11] + n \cdot 6 \tag{2}$$

The deadlines we have thus computed match exactly with the just-in-time schedule of Fig. 1(d), for all $n \geq 0$. If matrix $\mathbf{G}$ is the characteristic matrix of SDFG G, the matrix $\mathbf{G}^T$ is in fact the characteristic matrix of an SDFG, which we shall refer to as G^{-1}, obtained from G by reversing the direction of all edges. We have performed this operation on the SDGF of Fig. 1(a) to obtain the SDFG of Fig. 1(b). Eq. 1 suggests that there is a close connection between the just-in-time schedule of G and the self-timed schedule of G^{-1}; the latter is shown for the example in Fig. 1(e), assuming the starting vector $11 - [3\ 9\ 9\ 11]^T = [8\ 2\ 2\ 0]^T$ (where $[3\ 9\ 9\ 11]^T$ is the offset vector in Eq. 2 and 11 the maximum offset value). The following more formally investigates this relationship.

Proposition 3.1 *If SDFG G has the characteristic $(\max, +)$-matrix $\mathbf{G}$, then the reversed graph G^{-1} has the characteristic $(\max, +)$-matrix $\mathbf{G}^T$, the transposed of $\mathbf{G}$.*

Proof: $\mathbf{G}(k, m)$ is the length of the longest path (in sum of actor execution times on the path) from token m to token k in the homogeneous SDFG equivalent to G. If H is the homogeneous SDFG equivalent to G, then H^{-1} is the homogeneous equivalent of G^{-1}. (Details are left as an exercise to the reader.) Then, if $\mathbf{J}$ is the characteristic matrix of G^{-1}, then $\mathbf{J}(k, m)$ is the length of the longest path from m to k in H^{-1}, which is the length of the longest path from k to m in H and hence $\mathbf{J}(k, m) = \mathbf{G}(m, k)$, or $\mathbf{J} = \mathbf{G}^T$. ■

The following proposition intuitively states that $\boldsymbol{\gamma}$ is an in-time enabling condition for an SDFG with matrix $\mathbf{G}$ for deadline $K - \boldsymbol{\delta}$ ($\mathbf{G}\boldsymbol{\gamma} \preceq K - \boldsymbol{\delta}$) if and only if $\boldsymbol{\gamma}$ is at most equal to the result of a self-timed execution according to matrix $\mathbf{G}^T$ starting from vector $\boldsymbol{\delta}$, mirrored in time around time K ($\boldsymbol{\gamma} \preceq K - \mathbf{G}^T\boldsymbol{\delta}$). As an illustration, observe in Fig. 1 the mirror symmetry between the just-in-time schedule of G and the self-timed schedule of G^{-1}. Take for instance in the just-in-time schedule in Fig. 1(d) the first iteration ending in the vector of the encircled 1s: $[9\ 15\ 15\ 17]^T$. When we negate this vector and shift it by $K = 17$ to align it conveniently with time 0, we get the starting vector of the encircled 0s of Fig. 1(e): $[8\ 2\ 2\ 0]^T$. The actor firings to the left of the deadline vector $[9\ 15\ 15\ 17]^T$ in Fig. 1(d) are mirror images of the corresponding firings to the right of the starting vector $[8\ 2\ 2\ 0]^T$ in Fig. 1(e).

Proposition 3.2 *For any matrix $\mathbf{G}$, vectors $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ and scalar K: $\boldsymbol{\gamma} \preceq K - \mathbf{G}^T\boldsymbol{\delta}$ if and only if $\mathbf{G}\boldsymbol{\gamma} \preceq K - \boldsymbol{\delta}$.*

Proof: Because of linearity of the equations, we may assume w.l.o.g. that $K = 0$. We proceed to prove both directions independently. i ($\Rightarrow$) Let $\boldsymbol{\gamma} \preceq -\mathbf{G}^T\boldsymbol{\delta}$. We need to show that $\mathbf{G}\boldsymbol{\gamma} \preceq -\boldsymbol{\delta}$. By monotonicity, $\mathbf{G}\boldsymbol{\gamma} \preceq \mathbf{G}\left(-\mathbf{G}^T\boldsymbol{\delta}\right)$. Hence, it suffices to show that $\mathbf{G}\left(-\mathbf{G}^T\boldsymbol{\delta}\right) \preceq -\boldsymbol{\delta}$. We show this for each vector element n: $\left(\mathbf{G}\left(-\mathbf{G}^T\boldsymbol{\delta}\right)\right)(n) = \max_k \mathbf{G}(n,k) + \left(-\mathbf{G}^T\boldsymbol{\delta}\right)(k) = \max_k \mathbf{G}(n,k) + \left(-\max_m \mathbf{G}^T(k,m) + \boldsymbol{\delta}(m)\right) \leq \max_k \mathbf{G}(n,k) + \left(-(\mathbf{G}^T(k,n) + \boldsymbol{\delta}(n))\right) = -\boldsymbol{\delta}(n) + \max_k \left(\mathbf{G}(n,k) - \mathbf{G}^T(k,n)\right) = -\boldsymbol{\delta}(n) + \max_k \left(\mathbf{G}(n,k) - \mathbf{G}(n,k)\right) = -\boldsymbol{\delta}(n)$.

($\Leftarrow$) If $\boldsymbol{\gamma} \not\preceq -\mathbf{G}^T\boldsymbol{\delta}$, then there exists some m such that $\boldsymbol{\gamma}(m) > \left(-\mathbf{G}^T\boldsymbol{\delta}\right)(m) = -(\max_k \mathbf{G}^T(m,k) + \boldsymbol{\delta}(k))$. Let m be as such. Then $\boldsymbol{\gamma}(m) > -(\max_k \mathbf{G}(k,m) + \boldsymbol{\delta}(k))$. Let k be such that $\boldsymbol{\gamma}(m) > -\left(\mathbf{G}(k,m) + \boldsymbol{\delta}(k)\right)$. Then $\mathbf{G}(k,m) + \boldsymbol{\gamma}(m) > -\boldsymbol{\delta}(k)$ and hence $\max_n \mathbf{G}(k,n) + \boldsymbol{\gamma}(n) > -\boldsymbol{\delta}(k)$. Thus $(\mathbf{G}\boldsymbol{\gamma})(k) > -\boldsymbol{\delta}(k)$ and the required conclusion follows: $(\mathbf{G}\boldsymbol{\gamma}) \not\preceq -\boldsymbol{\delta}$. ■

Several interesting properties of self-timed and just-in-time schedules follow immediately from this result. The just-in-time schedule of an SDFG G is identical to the self-timed execution of the reversed graph G^{-1} when all start times are negated and when start and completion times are swapped.

Corollary 3.1 *Let $\{\boldsymbol{\delta}_k\}$ be the just-in-time schedule of an SDFG G to a deadline $\boldsymbol{d}(n)$ for n iterations. Let $\{\boldsymbol{\gamma}_k\}$ be the self-timed execution of the reversed graph G^{-1} starting from initial enabling conditions $\boldsymbol{\gamma}_0 = -\boldsymbol{d}(n)$. Then for all $0 \leq k \leq n$, $\boldsymbol{\delta}_k = -\boldsymbol{\gamma}_{n-k}$. The individual actor firings in both schedules coincide when all start and completion times are negated and when start and completion are swapped.*

For instance, for graph G of Fig. 1, assuming a deadline $\boldsymbol{d}(2) = [\infty\ \infty\ \infty\ 23]^T$ for $n = 2$ iterations, the just-in-time schedule of G, computed following Eq. 1 is $\{[3\ 9\ 9\ 11]^T, [9\ 15\ 15\ 17]^T, [\infty\ \infty\ \infty\ 23]^T\}$. Observe that this schedule corresponds to the schedule in Fig. 1(d), except for the last vector of the encircled 2s,

because in Fig. 1(d) it is assumed that the deadline on the Snk actor is repeated periodically, while in this example we consider only the deadline on Snk at $n = 2$. According to the relation expressed by the corollary, this schedule corresponds to the following self-timed schedule $\{[-\infty;\ -\infty;\ -\infty;\ -23]^T, [-9;\ -15;\ -15;\ -17]^T, [-3;\ -9;\ -9;\ -11]^T\}$ of G^{-1}. This schedule is conform the schedule of Fig. 1(e) when shifted by 23 time units to the right and similarly relaxing the initial conditions for actors different from the Snk actor to $-\infty$.

Corollary 3.2 *The just-in-time schedule of an SDFG G (like its self-timed schedule) is unique.*

Corollary 3.3 *The self-timed schedule is feasible if and only if the just-in-time schedule is feasible.*

Corollary 3.4 *The throughput of the self-timed schedule is equal to the throughput of the just-in-time schedule.*

The corollaries so far establish a means to compute a just-in-time schedule for an SDFG and they show the equivalence of self-timed and just-in-time execution in terms of feasibility and throughput. The following corollary, however, illustrates a difference between self-timed and just-in-time schedules when it comes to the sensitivity to occasional excesses of the actor execution time estimations.

Corollary 3.5 *Assume specific execution times of all actor firings in an SDFG execution. If the self-timed schedule of the SDFG violates a deadline, then the just-in-time schedule also violates that deadline. Conversely, however, there exist SDFGs with schedules such that the just-in-time schedule violates deadlines, while the self-timed schedule does not.*

As an example, look at the schedules of G in Fig. 1. In the just-in-time execution, any elongation of the first firing of B leads to the missing of the first deadline of Snk, due to its dependency on the completion of B. In the self-timed schedule, on the other hand, B may elongate its first firing by as much as 3 time units without violating the deadline of Snk.

IV. Experiment

We compare the robustness of self-timed scheduling and just-in-time scheduling to occasional excesses of the expected actor execution times. We have randomly generated 100 SDFGs with dedicated Src and Snk actors as in the running example, both firing once per iteration. Structure and token rates are determined randomly as well as initial tokens and actor execution times. The minimal period P is determined for each graph without the Src and Snk actors and the firing times of Src and Snk are set accordingly to P. We determine periodic deadlines for the Snk actor, setting the deadline for the n-th firing to $L + n \cdot P$ where L is determined to be 10% larger than the minimal achievable deadline (by self-timed execution). We perform with both the self-timed and the just-in-time schedules executions in which stochastic actor firing times are simulated. The firing time distributions are drawn from normal distributions with a mean of 90% of the original execution time and a standard deviation of 10%. In these simulated executions, an actor fires as soon as both its scheduled starting time has passed *and* its input dependencies are satisfied. For each graph we performed 500 runs of 500 iterations. From these simulated schedules we collected deadline miss statistics per graph.

TABLE I
Experimental results

schedule	*avg. % dl. misses*	*st.dev. % dl. misses*
self-timed	0.05%	0.13%
just-in-time	16.1%	2.20%

The results are summarized in Table I. For each of the schedules we show the average percentage of deadline misses, as well as the standard deviation in the percentage of deadline misses across the different random SDFGs. The results show that just-in-time scheduling is much more likely to lead to deadline misses compared to self-timed scheduling.

V. Conclusion

We have introduced a method to compute the unique just-in-time schedule for an SDFG with specified deadlines using the $(\max, +)$-model of SDF. The computation employs the transposed of the characteristic matrix of the SDFG. This result reveals a close relationship between the just-in-time schedule of an SDFG G and the self-timed execution of the reversed graph G^{-1}. We have further seen how a number of important conclusions follow from this result. Self-timed and just-in-time schedules perform equally in terms of feasibility and throughput, but they differ in terms of their robustness to the impact of exceeding the estimated execution times of tasks. Experimental results show that just-in-time scheduling is significantly less tolerant to execution time variations than self-timed scheduling. We believe it is straightforward to generalize the results of this paper to the more expressive workflow model of FSM-based SADF graphs [7], [2] by reversing not only the edges of the involved SDFGs but also the edges of the finite state machine.

References

[1] F. Baccelli et al. *Synchronization and Linearity*. Wiley, 1992.

[2] M.C.W. Geilen. Synchronous data flow scenarios. *ACM Trans. on Embedded Computing Systems*, 10(2): Article No. 16, 2010.

[3] A.H. Ghamarian et al. Throughput analysis of synchronous data flow graphs. In *Int. Conf. on Application of Concurrency to System Design, ACSD 06, Proc.*, p. 25–36. IEEE, 2006.

[4] B. Heidergott et al. *Max Plus at Work*. Princeton University Press, 2006.

[5] E.A. Lee, D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. on Computers*, 36(1):24–35, 1987.

[6] S. Sriram, S.S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*, 2nd edition. CRC Press, 2009.

[7] S. Stuijk et al. Scenario-Aware Dataflow: Modeling, analysis and implementation of dynamic applications. In *Int. Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation, IC-SAMOS 11, Proc.*, p. 404–411. IEEE, 2011.

[8] S. Stuijk et al. SDF^3: SDF For Free. In *Int. Conf. on Application of Concurrency to System Design, ACSD 06, Proc.*, p. 276–278. IEEE, 2006.